A low-power asynchronous data-path for a FIR filter bank

Nielsen, Lars Skovby; Sparsø, Jens

Published in:
Proceedings of the second International Symposium on Asynchronous Circuits and Systems

Link to article, DOI:
10.1109/ASYNC.1996.494451

Publication date:
1996

Document Version
Publisher's PDF, also known as Version of record

A Low-power Asynchronous Data-path for a FIR filter bank

Lars S. Nielsen[1)]

[1)] Department of Computer Science
Technical University of Denmark
DK-2800 Lyngby, Denmark

Jens Sparsø[1,2)]

[2)] Department of Computer Science
University of Utah
Salt Lake City, UT84112, USA

Abstract

This paper describes a number of design issues relating to the implementation of low-power asynchronous signal processing circuits. Specifically, the paper addresses the design of a dedicated processor structure that implements an audio FIR filter bank which is part of an industrial application. The algorithm requires a fixed number of steps and the moderate speed requirement allows a sequential implementation. The latter, in combination with a huge predominance of numerically small data values in the input data stream, is the key to a low-power asynchronous implementation. Power is minimized in two ways: by reducing the switching activity in the circuit, and by applying adaptive scaling of the supply voltage, in order to exploit the fact that the average case latency is 2-3 times better than the worst case. The paper reports on a study of properties of real life data, and discusses the implications it has on the choice of architecture, handshake-protocol, data-encoding, and circuit design. This includes a tagging scheme that divides the data-path into slices, and an asynchronous ripple carry adder that avoids a completion tree.

1 Introduction

Recent research has demonstrated that asynchronous circuit techniques have now matured and can be used to design integrated circuits with low power consumption - the most noteworthy examples being the DCC error corrector designed at Phillips Research Laboratories [1, 2] and the Amulet processors designed at Manchester University [3, 4].

Asynchronous circuits obtain their low power consumption for one or both of the following reasons:

- Circuits implementing algorithms whose computational complexity is data dependent enjoy a reduced switching activity because unused modules are not activated. Or to put it another way: No data latches are enabled unless there is new data to be stored in them. This reduced switching activity minimizes power consumption.
- If the typical/average computation takes less time than the worst-case computation, power consumption may be reduced by the use of adaptive voltage scaling [5]. A technique that converts excessive speed into a corresponding power saving.

The DCC chip takes advantage of both mechanisms: The number of steps in its Reed-Solomon algorithm is highly data dependent, and in the typical case entire sections of the algorithm may be skipped. This again allows the supply voltage to be reduced. The Amulet design exploits issues in instruction set processing.

Exploiting these mechanisms requires an experienced designer with a detailed understanding of the algorithm to be implemented as well as the data being processed by the circuit. Building up this base of experience and insight calls for more design experiments than the rather few reported up to now. The purpose of this paper is to contribute to this by considering a different application area that exhibits different optimization opportunities.

We are currently working on a low-power asynchronous implementation of an audio FIR filter bank that is part of an industrial battery powered application. Unlike the above mentioned designs, the filter algorithm does not exhibit any data dependent variations in the RTL level specification - the algorithm always requires the same fixed number of steps. Instead we exploit: (1) a highly non-uniform signal transition probability distribution (caused by a high correlation among input data), and (2) the fact that most data values have small magnitude. Both characteristics are found in many signal processing applications, and in combination with a highly sequential

0-8186-7298-6/96 $5.00

implementation, this makes it possible to design a low-power asynchronous circuit whose average speed is 2-3 times better than the worst case. Using adaptive scaling of the supply voltage, it is possible to convert this excess speed into a corresponding power saving. Details can be found in [5].

The paper is organized as follows. Section 2 describes the filter algorithm and the architecture used to implement it. Section 3 discusses characteristics that are exploited to minimize power consumption, and their implications on choice of communication protocol. Section 4 describes a number of implementation issues that contribute to minimizing power consumption. Section 5 demonstrates the speed and power advantages of the suggested architecture. Section 6 discusses the advantages of the asynchronous design and compares it to a synchronous, and finally, section 7 concludes the paper.

2 Algorithm and architecture

This section introduces the filter bank algorithm, motivates and describes the overall architecture of the circuit, and briefly outlines how the circuit can be embedded in an adaptive supply scaling environment.

2.1 Algorithm

The filter bank considered consists of a tree-like structure of interpolated linear phase FIR filters [6]. Explaining the details of the algorithm is beyond the scope of this paper. We only mention that much effort has been devoted to minimizing the number of multiplications, and to simplifying the multiplications by approximating the filter coefficients by numbers whose binary representation uses a minimum number of ones - a standard technique that significantly speeds up the multiplications. In this study we assume a maximum of 3 ones in the filter coefficients. Further more, a substantial number of the coefficients are zero and thus do not require an actual multiplication. Figure 1 shows a FIR filter with an additional complementary output, y_c. In the filter bank the two outputs are used to construct a binary tree structure. The outputs at the leaves of the tree delivers seven band-pass filtered versions of the input signal.

2.2 Architecture

The modest speed requirement of the application considered allows for highly sequential implementations. The algorithms can be serialized in several dimensions: using bit-serial arithmetic units and/or by serializing in the time domain by mapping the arithmetic units depicted in figure 1 onto a smaller set of hardware units.

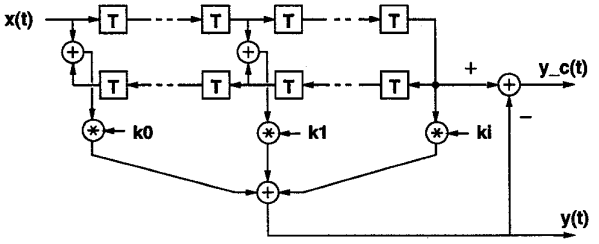

Figure 1: Interpolated linear phase FIR filter. The filter has two outputs, and the entire filter consist of a binary tree like structure of such FIR-blocks.

To avoid excessive power consumption due to handshaking overhead, bit-serial implementations should be avoided [7]. Also, structures where data is copied unchanged from one register to the next should be avoided. This means that a straight forward dataflow implementation with a hardware structure similar to the illustration in figure 1, should be avoided in practical/efficient implementations. This is especially the case when a large number of the coefficients are zero, because this requires a substantial amount of data shifting before the values are actually used.

These simple arguments hint that a processor like structure consisting of one or more memory blocks and one or more arithmetic units is the optimal choice. Figure 2 shows a structure that can implement the filter shown in figure 1, as well as the full binary tree structure we are currently designing.

All the delay elements (registers) in the binary tree filter structure are mapped onto a single dual-port RAM. The filter coefficients are stored in another RAM, and the computation is performed by a dedicated add-multiply-accumulate unit. Once an input data sample (or an intermediate result) is written into the RAM it stays in the same location. When time progresses one step and a new data sample is input to the filter, it is stored in the location that holds the oldest data sample (that is no longer needed).

The main task of the control unit is to generate the rather irregular sequence of read and write addresses that are needed. We do not discuss its implementation in this paper, it can be implemented in several ways. We only notice that it is possible to schedule the add-multiply-accumulate operations in such a way that a write to the memory from a FIR-block is not immediately followed by a read of the same location by some other FIR-block. If a pipelined implementation of the data-path is used, the pipeline would stall, waiting for the write to finish before the read could be performed. The absence of such tight loops allows the

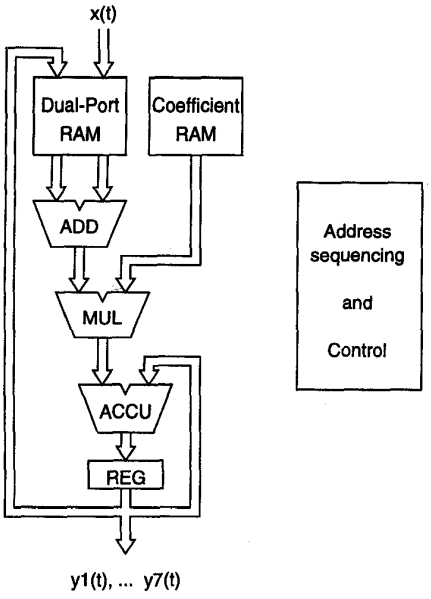

Figure 2: Architecture of the FIR filter bank processor.

control unit to be pipelined and to meet almost any speed requirement.

Also, the self-timed RAM is not described in this paper. We are currently studying a number of self-timed low-power register-file designs.

Finally, we cannot disclose exact figures for the filter bank that we are considering, but in order to provide some indication of the approximate size we mention that the filter bank calls for a RAM to hold several hundred data-samples. The number of coefficients are significantly smaller. The data-samples, the filter-coefficients and the internal busses are in the 10-20 bit range. The input is linear up to approximately 100dB sound pressure level.

2.3 Adaptive scaling of supply voltage

With the highly sequential implementation outlined above, variations in computation time due to data dependencies directly affect the total latency, i.e. the time it takes to process one input sample. Consequently the average case latency may be significantly smaller than the worst case. On the other hand the circuit must be designed for the worst case in order to cope with the fixed sampling rate.

A circuit of this nature is ideally suited for adaptive scaling of the supply voltage [5] - a technique that enables average "excess speed" to be converted into a corresponding power saving. In addition to data dependent variations in latency, this technique also exploits process variations and operating conditions.

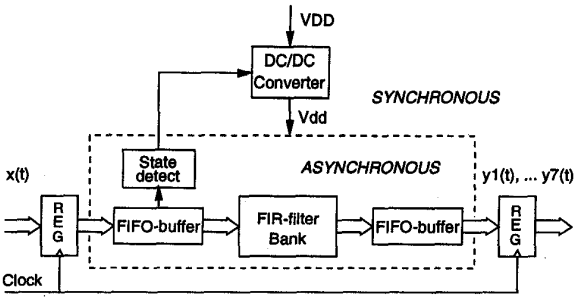

Figure 3: Self-timed circuit in synchronous environment using adaptive supply scaling.

The key idea is illustrated in figure 3 and briefly explained below. For more details the reader is referred to [5].

The system consists of the data processing circuit itself, two FIFO-buffers, a state detecting circuit, and a DC-DC converter for scaling down the supply voltage. The converter can be anything from a resistive device (a transistor on the chip) to a more sophisticated lossless device. Alternatively, the circuit may switch between different fixed supply voltages.

The state detecting circuit monitors the state of one of the buffers, for example, the input buffer as shown in Figure 3. If the buffer is running empty, the circuit is operating too fast and the supply voltage can be reduced. Similarly, if the buffer is running full, the supply voltage must be increased. In this way the supply voltage is adjusted to the lowest possible value that satisfies performance requirements.

3 Data dependencies

The input data stream to the filter is characterized by a huge predominance of small signal values as well as some correlation among the data samples. This means that the individual bits in a data-word have highly non-uniform switching probability. This section reports on an analysis of typical real life input data, and discusses the implications it has on the choice of number representation and communication protocol.

3.1 Characteristics of sampled input data

Figure 4 shows the signal transition probabilities in a five seconds recording of several people speaking at the same time, using a 17.5 KHz sampling rate, 16 bits resolution, and 2's complement representation. The figure shows a clear pattern that is typical in signal

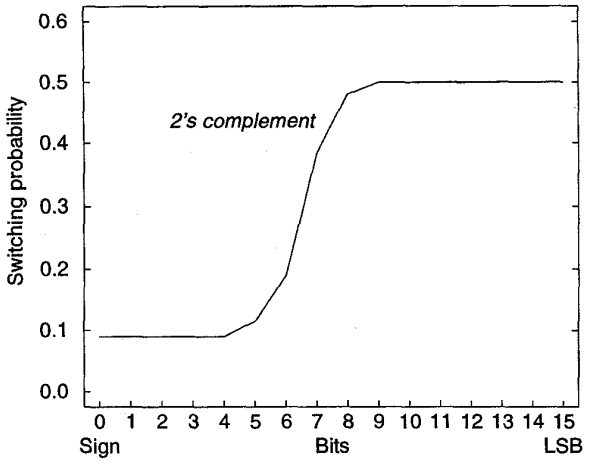

Figure 4: Switching activity profile of 5 seconds of sampled speech using 2's complement representation.

processing applications. The most significant bits 0 through 3 are outside the dynamic range of the signal and correspond to the sign and sign extension bits of the signal. These bits change whenever the sign of the data changes. Bits 8 to 15 are the least significant bits and they all have a 50% switching probability, which corresponds to uniform white noise. The rest of the bits correspond to the transition region between the least significant bits and the sign bits. The data here show that bits 0 through 3 can be discarded during processing, the information required is carried in bits 4 through 15. A switching profile like this is common to many signal processing applications and has been used by Landman and Rabaey to develop accurate high-level power estimation CAD-tools [8].

The analysis of switching activity shown in figure 4 is based on several people speaking at the same time for five seconds. However, for the application in question this is not the typical case. Most of the time the filter is idle, processing only background noise. Depending on the environment the background noise can have a number of different activity profiles, but common to most environments is that the sound pressure level is fairly low (otherwise we would not find them pleasant to be in). A sound pressure level around 40 dB is quite common.

A further analysis of switching activity shows that even during a normal conversation, the filter is idle, processing background noise for 20-40 percent of the time due to pauses in the conversation. In fact, the battery lifetime is dominated by the power consumed in the idle mode.

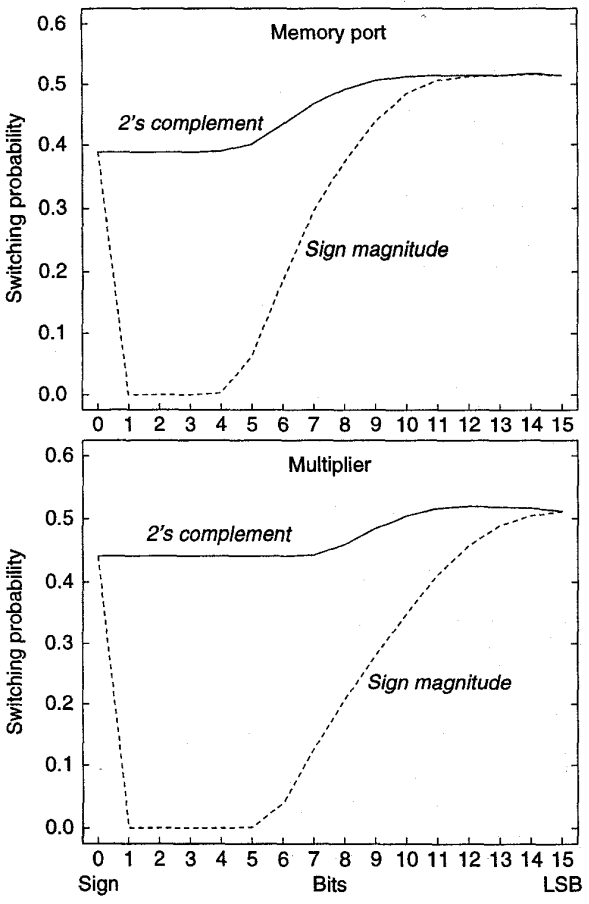

Figure 5: Switching activity profiles at the memory and multiplier output interfaces.

3.2 Number representation

The transition overhead of the sign bits shown in figure 4 is fairly small. The input values are highly correlated and the sign changes about each 10th time. But, these statistics are only valid for the input data. Inside the processing unit the activity profile is entirely different. Figure 5 shows the circuit activity at one of the memory output ports and at the multiplier output (the 16 most significant bits) when the data set displayed in figure 4 is applied. In both cases the profiles have been simulated using both a 2's complement representation and a sign magnitude representation. The upper part of the graphs shows the 2's complement and the lower part the sign magnitude.

From this figure it is obvious that the 2's complement representation has a much higher switching activity at module interfaces than the sign magnitude

representation. The overhead at the multiplier output is more than 100%, and as the dynamic range of the signal decreases the transition overhead can easily exceed 200%. In large circuits with heavily loaded busses, this overhead can have a significant impact on the power consumption of the circuit.

Choosing a sign magnitude representation instead reduces the interconnect power consumption, but power consumption inside the modules may increase. This is because a sign magnitude addition is a more complex operation to implement than a 2's complement addition. Adding two sign magnitude numbers, one positive and the other negative, may yield an intermediate negative result in 2's complement representation (involving a full sign extension). This intermediate result is then converted into sign-magnitude representation in a second addition (involving a full sign extension). For small numbers the transition overhead of the sign extension bits can be dominating. The choice of number representation is therefore not as obvious as figure 5 hints – both representations may lead to unnecessary switching activity on the most significant bits.

It was mentioned that most of the time the filter is in the idle state, during which only a small part of the bits actually carry important information. This suggests splitting the data-path into two or more slices and activating only the required parts of the data-path. In this way the transition overhead caused by sign bit extension can be minimized and at the same time the speed of the system can be increased. This can be implemented by augmenting the data words with a tag that indicates whether the full word is valid or only the bits corresponding to the least significant slice. Adders and other arithmetic units can use the tags associated with the operands to suppress switching activity (and carry propagation) in the most significant slice. The logic that deals with the tags is described in the next section.

The analysis of switching probabilities presented above shows that at least two operating modes can be identified: (1) processing of background noise, and (2) processing of actual sound. Slicing of the data path accordingly is one obvious solution. It might be worth dividing the processing of the actual sound into more than just one category, for instance, normal speech seldom amounts to more than 60 to 65dB. This suggests 3 operating modes: signals in 0 to 40dB range (background noise), signals in 40 to 65dB range (speech), and signals in 65dB to max range for all other types of sound.

It turns out that the add-multiply-accumulate data-path in the filter is dominated by additions. A 2's complement representation in combination with a sliced and tagged implementation is therefore chosen.

3.3 Handshake protocol and data encoding

Asynchronous circuits normally use one of the following three combinations of handshake protocol and data encoding: (1) 4-phase dual-rail (delay insensitive), (2) two-phase bundled data (micropipelines), and (3) 4-phase bundled data. Table 1 shows the number of wires and the number of signal transitions (including the req and ack signal wires) when communicating an N-bit data word from one module to another.

For the *bundled data protocols* the number of signal transitions depends on the transition probability of the individual bits. The worst-case value quoted in table 1 is when all bits have an uncorrelated switching probability $P = 0.5$.

For the 4-phase *dual-rail protocol* the number of signal transitions is independent of the switching probability of the data-bits. For every data-word transferred over the interface, N of the $2N$ data-wires make an upgoing transition followed by a down-going transition. This makes the switching activity 4 times larger than the *worst case* switching activity in the bundled data protocols.

Although the above simple arguments do not consider the switching activity inside circuit modules, it is fairly obvious that the 4-phase dual-rail protocol suffers from a significant transition overhead – four times larger than the worst case for the bundled data protocols. Also, it is not able to take advantage of the reduced switching activity found in many real life data as illustrated above. (Due to the slicing of the data-path this difference is less important in our design). The choice between the 4-phase and the 2-phase bundled data protocol is also a simple one. In our experience, register implementations for the 2-phase bundled data protocol are significantly larger or significantly slower than the ordinary latches that is used

Protocol	# wires	# transitions
4-phase dual-rail	$2N+1$	$2N+2$
2-phase bundled data	$N+2$	$< N/2+2$
4-phase bundled data	$N+2$	$< N/2+4$

Table 1: Simple comparison of asynchronous protocols.

in 4-phase designs. The same is true for control circuitry used to implement conditional sequencing. The reader may find more details and circuit level insight on these matters in [7]. Further more, if the decision is on precharge logic rather than static logic, then the four phase protocol comes as a natural choice: one handshake for the logic evaluation and one for the precharge operation.

The above is admittedly a simplistic picture, and because speed and power can be viewed as two sides of the same question, several protocols are often used in different places of a circuit. Our design is based on the 4-phase bundled data protocol, however, inside some modules the 4-phase dual-rail protocol is used (refer to section 4). This decision conforms with what seems to be a general trend when focus is on power and area (and possibly also speed): Philips Research Laboratories have re-targeted their Tangram Silicon Compiler from 4-phase dual-rail to 4-phase bundled data circuitry [2, 9], and the Amulet Group at Manchester University use 4-phase bundled data circuitry in the second version of their asynchronous ARM microprocessor (where the first version used 2-phase bundled data circuitry).

Finally we mention, that when 4-phase bundled data circuitry is used, the difference between synchronous and asynchronous data processing circuitry has diminished – asynchronous circuits can be viewed as synchronous circuits with a high degree of fine-grain clock gating, derived from the local request-acknowledge handshaking. There is one important difference however: asynchronous design techniques offer a systematic approach to obtain this fine-grain clock gating.

4 Implementation of the data-path

The previous section showed that sign extension can be very costly power wise. In this section we describe in detail the implementation of an add-multiply-accumulate data-path that takes advantage of the typical case dynamic range of the data. This includes slicing the data-path and suppressing most of the unnecessary sign extension activity in the most significant slice of the data-path. This scheme has the additional benefit that the circuitry computes faster when data with a small magnitude is input to the filter.

The term *break-point* is used to denote the borderline between the most significant slice and the least significant slice of the data-path, and terms like break-point adder and break-point multiplier are used to denote components operating with tagged operands and conditional activation of the most significant slice.

Op1-tag	Op2-tag	Res-tag
0	0	0/1
0	1	1
1	0	1
1	1	1

Table 2: Tag state table for an adder.

As this section shows, the data-path can be implemented entirely using adders. Special attention is therefore given to the efficient implementation of a self-timed break-point adder.

4.1 Tagging the operands

When a new data sample is input to the filter the value of its tag is computed and appended to the data word. If the MS part of the operand carries redundant sign extension information, the tag is set to 0, otherwise it is set to 1. As data flows down the data-path the magnitude of the operands may change, meaning that tag bits can change value as well. A full exploitation of the break-point concept therefore requires the modules to compute *both* the result and the associated tag. This represents a significant complication of the circuitry and a significant increase in power consumption.

Since all operands have zero tags in the typical case, we use a simple scheme where a module sets the result tag to 1 when one or more of its input operands have a nonzero tag or whenever an overflow occurs. More sophisticated schemes are not worthwhile, because they involve checking all bits above the break-point, and their higher complexity increases power consumption. With this simplification, the output tag state table for an adder is shown in table 2, leaving only the case where both input operands have zero tags unspecified.

For the case where both operands have zero tags we may do one of two things:

1. For the adder (marked ADD) in figure 2, we take advantage of the following observations: (a) an addition can only extend the result with one bit, (b) the adder is followed by a multiplier, and (c) all multiplications involve a filter coefficient in the range $]0; 0.5]$. On the output of the adder the break-point is therefore moved one position towards the most significant bit. After the multiplier the break-point is safely set back to the original position due to the third observation. The resulting and very simple tagging control logic for the add-multiply part of the data-path is shown

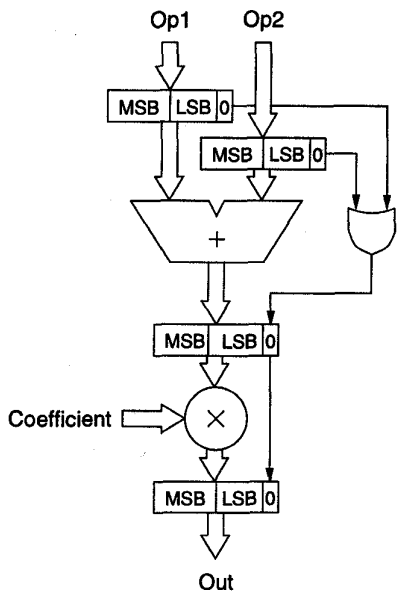

Figure 6: Tag control logic for ADD-MULT module.

in figure 6. The figure shows that only one OR-gate is required in the adder, and no circuitry is required in the multiplier.

2. A more general scheme, that is proposed for the accumulator, keeps the break-point in a fixed position. In the case where both input operands have zero tags, the result tag is set whenever an overflow occurs in the least significant slice.

4.2 A break-point adder.

The design of the break-point adder involves a tagging scheme and a carry completion scheme. These issues are addressed below.

4.2.1 The tagging scheme. The overall structure of a break-point adder implementing the more general tagging scheme is shown in figure 7. The adder has one break-point, which effectively divides it into two: Add_MS and Add_LS. Each of these adders have regular binary inputs and outputs, but the carry is represented using dual-rail encoding. Both adders use precharge logic. Add_LS is controlled directly by ReqAB, the request signal associated with the A and B operands. The request input to Add_MS is generated by the control circuit described below. To support this, Add_LS generates a dual-rail encoded overflow signal, Ow.

The TagCtl-circuit located between Add_MS and Add_LS in figure 7 generates a dual-rail encoded control signal, Ctl. Inputs to TagCtl are the tags of the operands (TagA and TagB), the overflow signal (Ow.t and Ow.f), and the input request signal, ReqAB. The true output, Ctl.t, is used directly as the result tag, TagSum, and it also indicates when to request/activate Add_MS. At the ReqSum output, a multiplexor determines which request to select based on the dual-rail Ctl signal. When Ctl is valid the MUX selects one of the inputs, otherwise the output is low.

The boolean equations implemented by the TagCtl circuit are:

$$Ctl.t = (TagA + TagB) \cdot ReqIn + Ow.t \quad (1)$$
$$Ctl.f = \overline{TagA} \cdot \overline{TagB} \cdot Ow.f \quad (2)$$

The MUX circuit implements the following boolean equation:

$$ReqSum = Ctl.t \cdot Req\_MS + Ctl.f \cdot Req\_LS \quad (3)$$

For completeness we also list the boolean equations for the overflow signals. In two's complement representation overflow occurs when the carry out of the most significant (sign) position is different from the carry into that position. If the most significant adder in Add_LS is denoted "m" and the carry "cy" the equations are:

$$Ow.t = cy_m.t \cdot cy_{m-1}.f + cy_m.f \cdot cy_{m-1}.t \quad (4)$$
$$Ow.f = cy_m.t \cdot cy_{m-1}.t + cy_m.f \cdot cy_{m-1}.f \quad (5)$$

In sign magnitude representation, overflow is simply the carry into the most significant (sign) position.

One situation is not accounted for in the above description of a two's complement implementation. When Add_MS is activated it is necessary to perform sign extension of operands with a 0 tag. For this reason the A_MS and B_MS inputs of Add_MS must be equipped with multiplexors that can select between the direct {A,B}_MS inputs or the sign extension of {A,B}_LS. The control signals, SelA and SelB, for these multiplexors are:

$$SelA = \overline{TagA} \cdot (TagB + Ow.t) \quad (6)$$
$$SelB = \overline{TagB} \cdot (TagA + Ow.t) \quad (7)$$

The circuitry represented by equations (1) to (7) constitutes the control overhead associated with the tagging scheme - a few small complex gates only. Furthermore, it should be noted that the sign extension circuitry represented by equations (6) and (7) does not consume power in the typical case, it is only activated

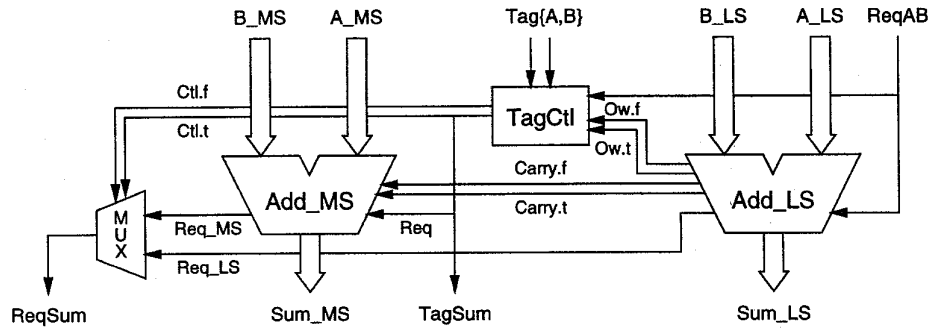

Figure 7: A self-timed break-point adder.

when the circuit is dealing with full length operands. With these observations we conclude that the power consumption of the overhead circuitry associated with the tagging scheme is negligible.

4.2.2 Completion detection. Because of the predominance of small data values and the serial implementation of the algorithm it is possible to exploit data-dependencies in carry propagation. For this reason a dual-rail carry signal is used. However, as the adder is of significant size, the speed (and power) penalty of a carry completion tree is likely to be significant. To avoid this, we suggest a hybrid scheme that avoids completion trees. A simple scheme is used in which the completion of an addition is indicated at the carry outputs of Add_MS or Add_LS depending on the input operands.

Figure 8 shows an N-bit adder using this scheme. In the design two full adder types are used, one that exploits the carry kill/generate states in the truth table, marked KG, and one that always waits for all of its operands, marked P (propagate). The adder works as follows: If FA(N/2) can generate a carry output without waiting for its incoming carry, this carry is generated, and ripples/propagates through all the more significant adders and eventually C_{out} becomes valid. This signals the end of the computation. Assuming equal delay in the two adder types, the delay through adders FA(N/2) up to FA(N-1) matches or exceeds any carry propagation delay in adders FA(0) to FA(N/2-1), and the correct operation of the adder is therefore ensured. In this way the carry propagation delay in the entire adder ranges from N/2 (in 50 % of the cases) up to N. Add_LS is implemented in this way.

The same principle is applied again to the entire adder, consisting of Add_MS and Add_LS. This means that Add_MS is similar to the upper half of the adder in figure 8. Therefore, when the magnitude of the data is above the break-point, the computation time ranges from 50% to 100% of the worst computation time. When data is below the break-point the computation time ranges from 25% to 50%.

The break-point solution suggested here is a simple but effective one when most data have a small magnitude, as in our case. Other more complex break-point schemes can be used to gain a better speed (which can be traded for power) but at the expense of more circuitry. The best trade off can only be determined after extensive investigations, but in many cases it turns out that the better solution is the simplest one.

4.3 A break-point multiplier

It was mentioned previously, that the filter coefficients are approximated with values whose binary representation contains at most three 1's. This significantly simplifies the multipliers, resulting in smaller area and higher speed. Figure 9 shows a possible implementation which is both small, fast, and has a data dependent computation time. The coefficients have been replaced by the control signals C1-C3 that control the input shifters and Sel which controls the output multiplexer.

The adders framed by the dotted line are connected in such a way that the second adder starts computation immediately after the first bit has been computed in the first adder. This gives a computation time close to one addition, however, a full length carry propagation is required in the Add_LS part of the second adder. The multiplier has been further optimized for coefficients containing only one 1 (which frequently occurs in the present application) by adding a multiplexer at the multiplier output. In this case the additions can be skipped entirely, thus saving transitions and speeding up the computation.

4.4 A break-point accumulator

The accumulator is simply a break-point adder with a feed back loop. The main concern with the accumu-

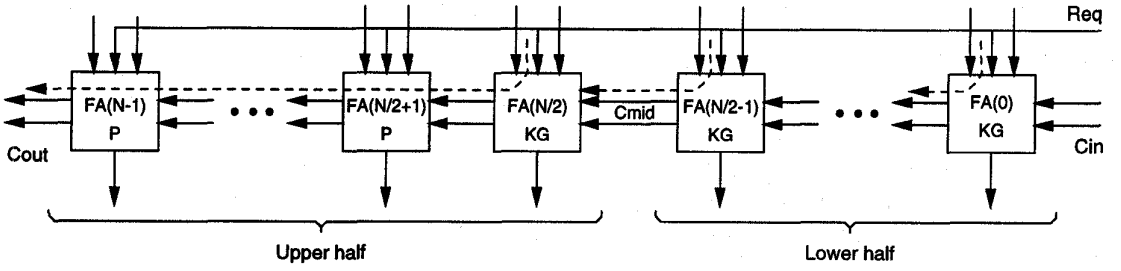

Figure 8: Carry propagation scheme (used in Add_LS)

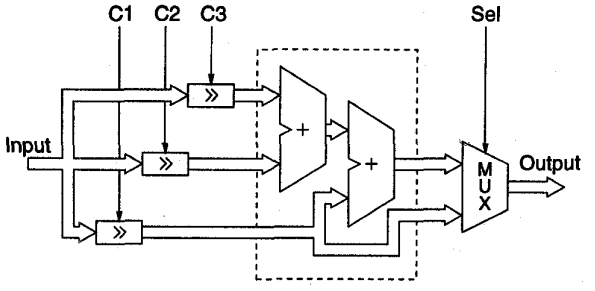

Figure 9: Self-timed break-point multiplier

lator is: will the magnitude of the accumulated value be larger than the break-point value. However, looking at the frequent sign change of the operands at the multiplier output (refer to figure 5), it is highly likely that the magnitude of the accumulated value does not change that much.

Further simulations confirm this theory – simulating the switching probability in the accumulator gives a probability profile almost identical to that of the memory output port shown in figure 5.

5 Performance evaluation

To demonstrate the performance of the architecture presented, a 16 bit filter design is evaluated. The design is assumed to have four extra bits in the accumulator, and 30% of the coefficients are simple shift operations (the numbers have close resemblance to the application considered). Each pass through the data-path requires a computation time equal to the sum of each of the three modules in the data-path. If no pipelining is applied, the total computation time per data sample is determined by the number of iterations required, n:

$$t_{sample} = \sum_{i=1}^{n} t_{add} + t_{multiply} + t_{accumulate} \quad (8)$$

In the following analysis we assume that n is high. In that case the total computation time t_{sample} approaches the sum of the average computation time of each of the modules. With these assumptions a statistical analysis of the filter gives the results in table 3. The analysis does not include the overhead of the handshake control circuitry, neither does it include the delay in the multiplier shifters and multiplexer. The adder worst case computation time of the 16 bit input adder is thus 16Δ, where Δ is the delay of one adder. In the fastest case data only propagates 4 places, and in the average case carry propagates 4.8 places (the average case corresponds to processing of background noise). Due to the switching probability profile of the input data, the average performance is very close to the best performance. Summing up the statics of each of the modules shows that the average performance of the data-path is $56/18.6 = 3.0$ times faster than the worst computation time of this architecture. It might be worth considering a pipelined solution to increase the speed of the system and lower the supply voltage even further. However, the speedup will not be proportional to the degree of pipelining – one of the stages is likely to constitute a bottleneck. Which stage may vary due to data dependent variations in the latency of the stages. This argument suggests that the total computation time per data sample, assuming a 3 stage pipeline, can be approximated by the following

Module	Worst case	Best case	Av. case
Adder	16Δ	4Δ	4.8Δ
Multiplier	20Δ	0Δ	7.8Δ
Accumulator	20Δ	5Δ	6Δ
Filter	56Δ	9Δ	18.6Δ

Table 3: Estimated computation time of the filter

equation:

$$t_{sample} = \sum_{i=1}^{n} max\{t_{add}, t_{multiply}, t_{accumulate}\} \quad (9)$$

This shows that the gain in speed will be moderate. A factor of 1.5 rather than the expected factor of 3 is a good estimate. Also, both the handshaking overhead and the number of signal transitions in the design increases due to the latches introduced. It therefore requires a careful analysis to determine whether or not the extra speed can be traded for power by further scaling of the supply voltage.

The power savings that can be obtained, depends on the supply voltage of the system, V_{DD}. For large values of V_{DD}, the circuit speed scales linearly with V_{DD}, but as V_{DD} approaches two times the transistor threshold voltage V_{th}, the circuit speed slows down dramatically [5]. In a standard 1 micron CMOS process with V_{DD}=5V, a factor of three typically makes it possible to halve (or more) the supply voltage, which in the best case reduces the dynamic power consumption by a factor of four (not considering short circuit currents and velocity saturation which makes it even more attractive [5]).

The power consumption also depends on the switching activity in the data-path. Assuming a two's complement representation the switching activity inside the data-path is close to 50% (c.f. figure 5) and therefore the power reduction is almost proportional to the slicing of the data-path. Splitting the data-path into two slices with identical width as in the example, nearly halves the power consumption in the data-path.

The combined effect of reduced switching activity and scaling of the supply voltage, as discussed above, reduces power consumption by a factor of 8. Even though no absolute estimates of the power consumption are available at this early stage, this significant factor is more than enough to justify the design.

6 Discussion

Comparing the architecture presented in this paper with a synchronous architecture, the handshaking overhead and the extra logic needed for slicing of the data-path has to be considered.

If the asynchronous data-path is implemented without pipelining (as we propose), the overhead of the handshaking is minimal. With the bundled data protocol it is only one C-element per stage (adder, multiplier or accumulator) in the data-path. To gain a speed-up in a synchronous implementation, similar to that of the non-pipelined asynchronous solution, it is necessary to use pipelining or carry look ahead arithmetic, and both techniques represents a significant overhead in terms of area and power.

If pipelining was to be used in the asynchronous data-path, the speed penalty of the handshaking is likely to increase. Without pipelining only one of the modules is active at a time, and the inactive modules have plenty of time to return to the initial state before the next computation. With pipelining, the reset phase of the handshake is likely to enter the critical path, and limit the performance gain. Considering the area and power overhead, it is therefore unlikely that pipelining of the asynchronous data-path will pay of.

The proposed slicing of the data-path could also be used in a synchronous design, but only as a means to reduce the switching activity. The associated speed advantage can not easily be exploited. The synchronous equivalent to what we are doing would be to vary the period of the clock signal, which is much less feasible than clock gating.

The control circuitry needed to slice the data-path (described in section 4) does affect the latency of the data-path. However, in view of the significant gain in average case performance, this is not an issue. Also, it should be noted that almost the same circuitry would be needed in a synchronous implementation, and in that sense it does not constitute an overhead.

In summary the non-pipelined asynchronous implementation has a number of unique advantages, and its circuit overhead is negligible.

7 Conclusion

This paper has described a number of issues relating to the design of a low-power asynchronous FIR filter block. Like many other signal processing applications, this algorithm does not exhibit data dependencies at the RTL level – the number of steps is fixed. Instead the key to a low-power implementation lies in a highly non-uniform switching profile of the data that is processed – something that is also common in signal processing applications.

The paper has showed by example, how this can be exploited to obtain an implementation in which the switching activity is minimized and the speed is maximized by taking advantage of data dependent computation times in the functional units. In our case the typical speed is 3 times better than the worst case, and using adaptive scaling of the supply voltage, this excess speed can be turned into a corresponding (additional) power saving.

Another important point to make is that a synchronous implementation cannot exploit these data

dependencies using clock gating. The equivalent to what we are doing would be to vary the period of the clock signal, which is much less feasible than clock gating.

Circuit design is ongoing and the ultimate goal is a speed and power comparison with an industrial synchronous design (fabricated on the same wafer). The design has two challenging areas, besides the datapath reported in this paper: Design of a low-power memory/register file, and design of the addressing and control unit. Work on these issues is ongoing.

References

[1] C. H. van Berkel, Ronan Burgess, Joep Kessels, Ad Peeters, Marly Roncken, and Frits Schalij. Asynchronous Circuits for Low Power: a DCC Error Corrector. *IEEE Design & Test*, 11(2):22–32, 1994.

[2] Kees van Berkel, Ronan Burgess, Joep Kessels, Ad Peeters, Marly Roncken, Frits Schalij, and Rik van de Viel. A single-rail re-implementation of a dcc error detector using a generic standard-cell library. In *2nd Working Conference on AsynAsynchronous Design Methodologies, London, May 30-31, 1995*, pages 72–79, 1995.

[3] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, S. Temple, and J. V. Woods. The design and evaluation of an asynchronous microprocessor. In *Proc. Int'l. Conf. Computer Design*, October 1994.

[4] S. Furber. Computing without clocks: Micropipelining the ARM processor. In G. Birtwistle and A. Davis, editors, *Proceedings Banff VIII Workshop: Asynchronous Digital Circuit Design*, Workshops in Computing Science, pages 211–262. Springer-Verlag, 1995.

[5] L. S. Nielsen, C. Niessen, J. Sparsø, and C. H. van Berkel. Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. *IEEE Transactions on VLSI Systems*, 2(4):391–397, 1994.

[6] T. Lunner and J. Hellgren. A digital filterbank hearing aid – design, implementation and evaluation. In *Proceedings of ICASSP'91*, Toronto, Canada, 1991.

[7] Jens Sparsø, Christian D. Nielsen, Lars S. Nielsen, and Jørgen Staunstrup. Design of self-timed multipliers: A comparison. In S. Furber and M. Edwards, editors, *Proc. of IFIP TC10/WG10.5 Working Conference on Asynchronous Design Methodologies, Manchester, England, 31 March – 2 April 1993*, pages 165–180. Elsevier Science Publishers B. V. (IFIP Transactions, vol. A-28), July 1993.

[8] P. Landman and J. Rabaey. Architectural power analysis: The dual bit type method. *IEEE Transactions on VLSI Systems*, 3(2):173–187, 1995.

[9] Ad Peeters and Kees van Berkel. Single-rail handshake circuits. In *2nd Working Conference on Asynchronous Design Methodologies, London, May 30-31, 1995*, pages 53–62, 1995.